MEASURING SOFTWARE ENGINEERING

Ciara Moynagh

17340087

Table of Contents

This report considers the ways in which the software engineering process can be measured and assessed in terms of measurable data, an overview of the computational platforms available to perform this work, the algorithmic approaches available, and the ethics concerns surrounding this kind of analytics.

Introduction

Managers are always looking to improve productivity. Collecting software metrics may be an important way to help identify, prioritise, track and communicate the current areas of downfall within a software development team. The goal of tracking and analysing the software engineering process is to understand the quality of the process which is currently being used, to improve this quality and to predict the quality that will be output if we work to enhance these metrics.

However, software engineering can be a complex profession to track. Unlike many other engineering disciplines, its output is not physical. It is instead the inception and implementation of an intangible software system. Because of this, it can be difficult to measure the process, and it is often that redundant- or even anti-proactive- measurements are taken into account. Because of this, it is vital to use measurable data that is linked to team goals and objectives when measuring and assessing the software engineering process.

Take lines of code count, for example. As a gauge of productivity, this measurement is out-dated and nonsensical. Workers are likely to approach their work in favour of meeting requirements- overcomplicating code by adding more lines than is necessary- instead of aiming to provide the best product for the end user. To quote Israeli business management guru Eliyahu Goldratt, “Tell me how you measure me, and I will tell you how I behave”.

The Ways in which the Software Engineering Process can be Measured and Assessed in Terms of Measurable Data

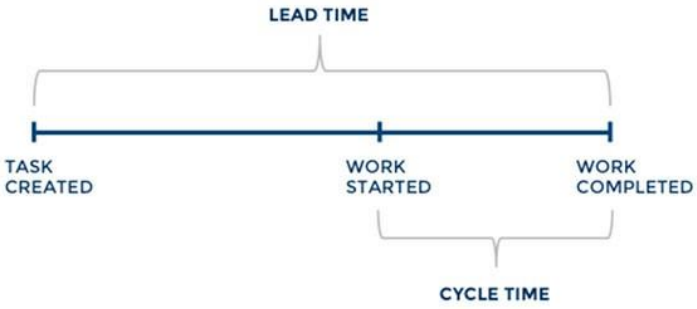
The understanding that to optimise productivity, collected data must relate to team goals, has brought about a set of software metrics known as the Agile Software Engineering Metrics. Looking from an agile point of view, there are three important families of metrics: these being Lean metrics, Kanban metrics, and Scrum metrics (Sealights, 2019).

Lean Metrics

Lean measurements focus on ensuring a steady flow of value from the team/organisation to their customers. They also focus on eliminating any wasteful activity. Such metrics include *lead time* and *cycle time*.

Lead Time

Lead time takes a measurement of the total time taken from the moment a story enters the backlog of the system, until it is completed or released to the customer. It measures the time taken before a requirement begins to earn value- the speed of the entire agile system value chain from end to end. Reducing lead time results in a more efficient development pipeline.



Cycle Time

Cycle time is a subset of lead time, as is represented in the above diagram. It measures the time taken for a task to move from the 'start' point, to 'finished'. It should be approximately half the length of the sprint. It is a very simple metric, which raises a clear red flag if teams are not completing the work they have committed to, and the cycle time is longer than the sprint.

Kanban Metrics

Kanban metrics focus on the organisation and prioritisation of the workflow. A *cumulative flow metric* is a well-known form of Kanban metric.

Cumulative Flow Diagram

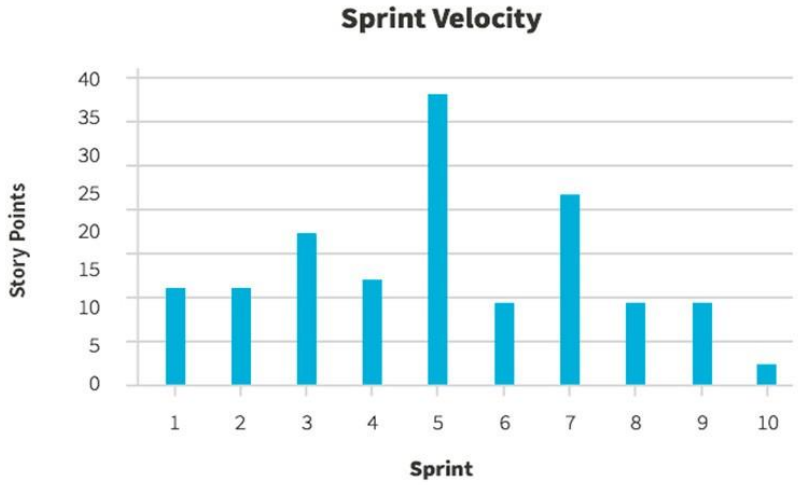
This Kanban metric shows the status of a task to the team at a glance. It is a time-based plot which is typically plotted on a daily basis, visualising whether there is a disproportionately large number of tasks at any stage of the workflow. The various colour bands indicate the different stages of the workflow, and the height of each band at a specific point in time indicates the number of cards in that stage. The power of this metric is in its simplicity. A team can understand their progress and comprehend any obvious issues at a glance.

Scrum Metrics

Scrum metrics focus on the predictable delivery of working software to customers. Common types include *team velocity* and the *burndown chart*.

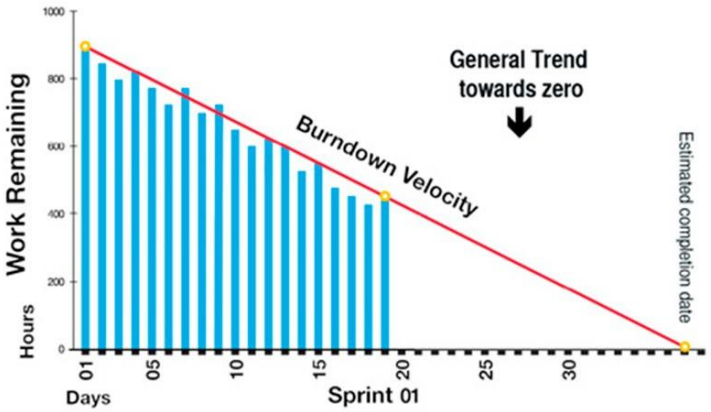
Team Velocity

Team velocity measures the average amount of software units (aka. story points) that a team completes in one iteration or sprint. Essentially, it measures how fast your team is moving. This metric cannot be used to compare different software teams, as the definition of deliverables will be completely different from one team to the next. It is used as an internal metric within each individual team. It is a result based metric- representing how much value was actually delivered to customers in a series of sprints. It can be used to predict the team's output in upcoming sprints.



Sprint Burndown Charts

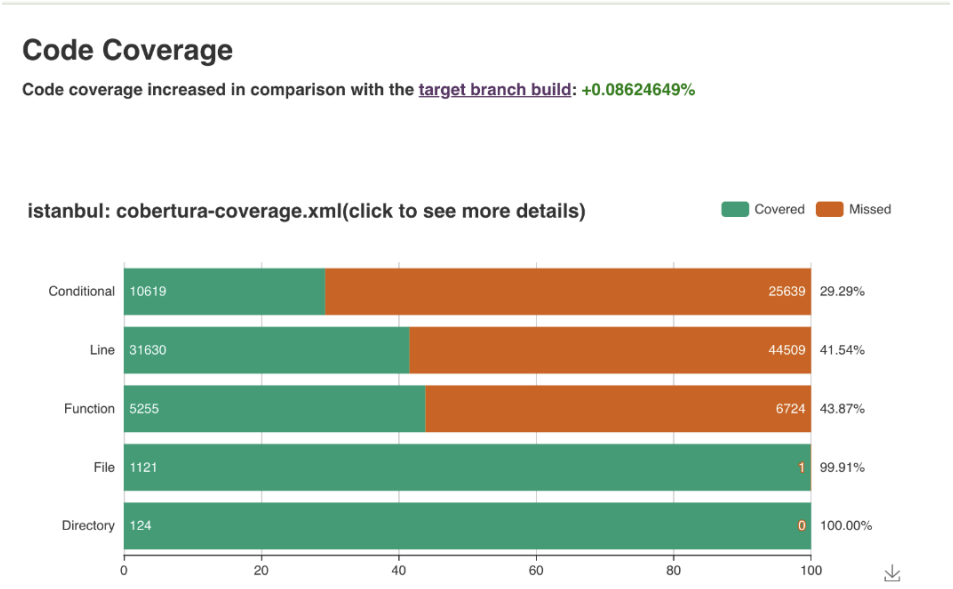
A sprint burndown chart gives a visualisation of the amount of story points which have been completed during a sprint. It presents a visualisation of the amount which remain, and helps to forecast if the scope of the sprint can be completed on time. It is a powerful form of representing metrics as it instantly conceptualises the value that a sprint has delivered thus far, and how much further work remains.



Further common agile metrics include:

Code Coverage

A measurement of the percentage of code covered by unit tests is known as code coverage. A measurement of the number of methods, branches, conditions or statements that are executed as part of a unit test suit is evaluated. It is a useful measurement as it can be run automatically each time the code is built, making it quick and easy to appraise. Low code coverage will almost always indicate dissatisfactory code.

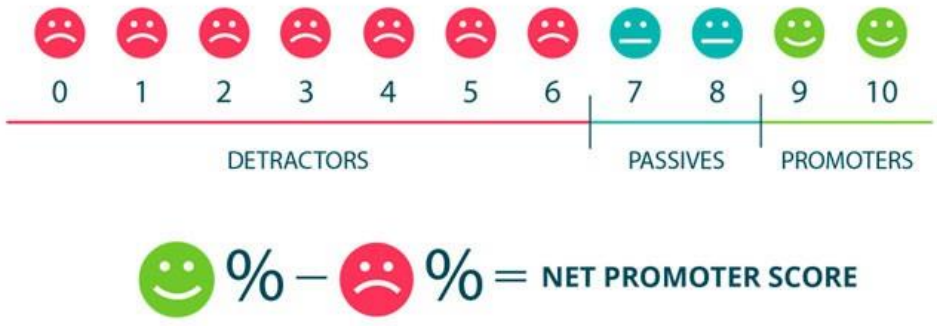


Static Code Analysis

This is another automated process which takes simple error redundancies to provide insight into code quality. While not exactly a metric, it is known to be a key contributor to the general quality of the development process, and into the maintainability of the software.

Release Net Promoter Score

The Net Promoter Score measures whether a user would recommend the software to others. It is an important metric in gauging whether value has been provided to the customer- the ultimate test of agile development. If customers would not recommend the new release to others, this can be used as a warning metric. Other data can then be applied to understand where the problems lie.



Failed Deployment Rate

Otherwise referred to as the mean time to failure, this metric determines how often developments prompt outages or other issues. This number should be as low as possible. An increasing failed deployment rate probably suggests dysfunction somewhere along the workflow. This can help us to understand whether teams are really building functional software.

Escaped Defects

This measurement counts the number of bugs discovered in the code after the software has been released/enters production. Ideally, this number should be zero. When measuring this number across teams or releases, we can evaluate a highly relevant measurement of the quality of software being deployed.

Happiness

The reason Agile metrics have made such an innovative impact on the way in which the software engineering process is measured is because they attempt to take data such as employee satisfaction into account. There is no single, tried, tested and proven way to measure happiness in a software development team. However, a common solution is to ask every team member to rate their current level of happiness on a scale of 1 to 5. According to organisational psychologist Christiaan Verwijs, this can be followed by several questions, such as:

- How happy are you with your company?
- What feels best and worst right now?
- What would help to increase your happiness?

Team Morale

Varwijs proposes that instead of measuring happiness- as this metric can be unreliable due to its subjectivity- management could measure team morale. Measuring team morale is a more subtle, team and task-oriented way, resistant to changes in mood. According to Varwijs, measuring team morale should be assessed/approached by asking team members to rate the following statements on a scale of 1 to 5:

- I feel fit and strong in my team
- I am proud of the work that I do for my team
- I am enthusiastic about the work that I do for my team
- I find the work that I do for my team meaningful and purposeful

These questions are more specific, allowing management to better access the well-being of team members.

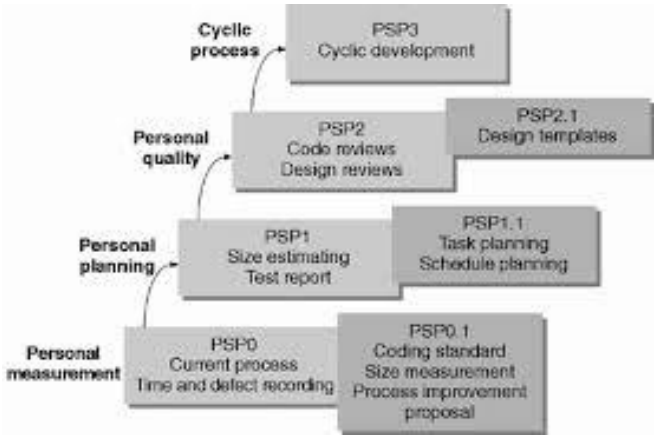
An Overview of the Computational Platforms Available to Perform this Work

There are various tools and programs available to measure the software engineering process. Which computational platform a manager puts in place ultimately depends on the work being done and the way in which management wishes to measure this work. Below I have provided an extensive overview of a few examples.

Personal Software Process

The Personal Software Process (PSP) was introduced by Watts Humphry, and was one of the first methodologies designed for the computation of a software engineers performance. It allows engineers to measure their work by manually recording and analysing various simple metrics in a spreadsheet format, providing developers with a framework for best practice and self-evaluation throughout the software engineering process. PSP works on the basis that once a certain level of increased productivity has been reached, engineers must then strive to improve their process once again. There is a set list of steps which must be taken to increase performance, according to PSP:

1. Define the quality goal
2. Measure the product quality
3. Understand the process
4. Adjust the process
5. Use the adjusted process
6. Measure the result
7. Compare the result with the goal
8. Recycle and continue improving



GitPrime

GitPrime is a paid professional organisational tool, providing a platform from which to measure and monitor the software development process. It provides team leaders with contextual metrics, such as the percentage of new code in each commit, how much of the team's burn is going into refactoring old code, whether the change in the sprint cycle had a net-positive effect, etc. GitPrime can be used to determine which commits were riskiest to the development of the overall process, which aids in pointing out the commits that should be reviewed in depth by the entire team. The overall aim of GitPrime is to increase visibility about team contributions, find the areas where the biggest impact is being made, identify areas in need of improvement, and help the team understand how changes in the process can impact the team's productivity and efficiency.

(Screenshot from GitPrime)

Humanyze

If measuring the process alone has not provided satisfactory results with improving team productivity, Humanyze- founded in 2010 in Boston, Massachusetts- may provide the answer. Humanyze is an employee analytics software provider, which aims to assist companies in measuring corporate communication data to uncover patterns in the manner in which work is getting done. They offer 'Sociometric Badges'- a sort of sensor to be worn by each employee. The purpose of these is to track and capture everything that is going on throughout the day within the organisation.

Each sensor contains a microphone to record speech, Bluetooth to track location, and infrared to detect with whom each employee is speaking to. Humanyze have a strict data protection policy, ensuring that all data is kept anonymous and the information gathered is being used for nothing other than analysis of the overall productivity measures.

An Overview of the Algorithmic Approaches Available to Perform this Work

Cyclomatic Complexity

Cyclomatic complexity (CYC) is an algorithmic software metric used to indicate the complexity of a program. Introduced by Thomas J. McCabe in 1976, it quantitatively measures the number of linearly independent paths throughout a program's source code. To compute the cyclomatic complexity, the control flow path of a program is used. The nodes of any such graph correspond to indivisible groups of commands within a program, and the directed edges connect any two nodes such that the second command might be executed immediately after the first.

For example, if the source code were to contain no control flow statements, the complexity would be 1, as there would be only a single path through the code. If the code had one single-condition IF statement, there would be two paths through the code: one where the IF statement evaluates to TRUE and another one where it evaluates to FALSE, so the complexity would be 2. Two nested single-condition IFs, or one IF with two conditions, would produce a complexity of 3. The lower the

complexity, the better the code. This is because code with high complexity is difficult to test, and more likely to result in errors.

Cyclomatic complexity may also be applied to individual functions, modules, classes and methods within a program.

How to calculate Cyclomatic Complexity:

The following formula is used when calculating CYC:

CYC = E - N + 2P

Where:

- E = number of edges (transfers of control)
- N = number of nodes (sequential group of statements containing only one transfer of control)
- P = number of disconnected parts of the flow graph (such as a calling program and a subroutine)

Halstead Software Science Complexity

Halstead complexity measures were developed by Maurice Howard Halstead in 1977- the idea being that software metrics should reflect the expression or implementation of algorithms in different languages, but at the same time, should be independent of their execution on a particular platform. Therefore, these metrics are computed statically from the code.

How to calculate Halstead Complexity measures:

For a given problem, let:

- n1 = the number of distinct operators
- n2 = the number of distinct operands
- N1 = the total number of operators
- N2 = the total number of operands

A number of measures can be calculated using these numbers:

- Program vocabulary: n = n1 + n2
- Program length: N = N1 + N2
- Calculated estimated program length: N' = n1 log2 (n1) + n2 log2 (n2)
- Volume: V = N * log2 n
- Difficulty: D = (n1/2) * (N2/n2)
- Effort: E = D * V
- Time required to program: T = E/18 seconds
- Number of delivered bugs: B = V/3000

Algorithm Cost Modelling

Algorithmic cost modelling uses a mathematical expression to predict project costs based on estimates of the project size, the number of software engineers, and other process and product

factors. An algorithmic cost model can be developed by analysing the costs and attributes of completed projects and finding the closest fit mathematical expression to the actual project.

How to calculate the Algorithm Cost Model:

Effort = A * S^B * M

Where:

- A = a constant factor, dependent on local organisational practises and the type of software being produced
- S = a variable which represents either the code size or the functionality of software expressed in object or function points
- B = an exponential associated with the size estimate, expresses the non-linearity of costs with project side (value usually lies between 1 and 1.5)
- M: a multiplier, made up of process, product and development attributes, such as the dependability requirements for the software experience of the development team

Bayesian Belief Networks

A good example of a machine learning algorithm is the Bayesian belief networks. These are a collection of classification algorithms based on Bayes' Theorem, and consisting of three basic components. The first is a measurement of a network structure and a database, the second is a search heuristic that chooses network structures to be considered, and the third is a method of estimating the probability tables from the database.

A Bayesian belief network consists of a directed acyclic graph together with an associated set of probability tables. The graph is made up of nodes which represent discrete or continuous variables, and arcs which represent causal relationships between these variables. The Bayesian Network provides a means of defining a probabilistic model for a complex problem by stating all the conditional independence assumptions for the known variables, whilst also allowing the presence of unknown variables. As such, both the presence and the absence of edges in the graphical model are important in the interpretation of the model.

For example, if we are to enter evidence regarding the number of defects found when testing, all the probabilities in the table are updated. The probability of every state will always be computed, regardless of the amount of evidence provided. Lack of evidence is reflected with greater uncertainty values.

The success of this method is reliant on the stability and maturity of the development process. Organisations must collect basic metric data and carry out systematic testing in order to apply this model effectively.

Notable software to calculate Bayesian networks include PyMC3 (A Python library implementing an embedded domain specific language to represent Bayesian networks), SPSS Modeler, and OpenBUGS.

The Ethical Concerns Surrounding these kinds of Analytics

The monitoring of individuals within any work environment raises much discussion and concern regarding ethicality and morality. In today's world, employees are becoming increasingly sensitive and intolerant towards unethical practice towards the privacy of their data. Business owners must be careful not to overstep boundaries which could drive their employees away. The key trade-off when conducting analysis of the software engineering process is between obtaining rich and insightful information, and impeding on software developers rights to privacy.

Employers must ensure that all employees receive full transparency regarding the data collection policy that is in place. Developers must first of all know and understand that they are being monitored and measured. Employers must also disclose what kinds of metrics are being taken and recorded, to reduce employee discomfort. For example, take a look at Hackystat, as discussed above. When using this platform, an extensive range of measurements are being recorded at any given moment. This has been seen to cause some employee discomfort, as employees find it impossible to understand every manner in which they are being measured. The software has been known to be referred to as 'Hacky-stalk' by employees (Johnson, 2013).

Access to the information gathered and measurements taken must also be limited. Many employees are not going to be comfortable with their sensitive information being widely available, especially if this information contains any personally identifiable data. In addition to this, each individual employee should have access to all metrics recorded on them, personally. This will lead to a more trusting and transparent workplace, and may also help in motivating employees to improve on these metrics.

A final ethical concern that requires attention from employers is that of the way in which productivity is evaluated. Software engineering is a very complex profession, so using quantitative measures alone to measure the process cannot give a fair, well rounded depiction of the procedure. To reduce the entire process down to a few simple figures is unfair on employees and should ultimately be avoided. How well an employee adds to staff morale, fits in with the company culture, encourages, motivates and assists fellow members of staff, provides new, innovative ideas etc. should all be monitored and rewarded too. Because of this, Agile metrics provide a more fair and equal assessment.

Conclusion

It is evident that there are plenty of mechanisms, platforms and algorithms available to help monitor and assess the software engineering process. I believe that the measurement of this process is inherently complicated, but can add valuable and realism by use of Agile methodology metrics.

The ability to measure the software engineering process is convenient and favourable from a management point of view, as it helps to reduce a very complex process down to a few simple facts and figures. However, management must not ignore the fact that current software engineering measurements do not take some of the most important points of potential added value into account- as these complex areas cannot be quantified.

Although advantageous for management to gather such metrics, a balance must be struck between gaining quality insights, and disrupting employee privacy. Failing to do so may arise in a major impact on worker trust and productivity, which far outweighs the benefits of collecting these metrics. By use of Agile methodologies, we can help to combat this problem.

Bibliography

Fenton, N. and Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, 47(2-3), pp.149-157. [Accessed 24 Oct. 2019]

Sealights. (2019). *10 Powerful Agile Performance Metrics - and 1 Missing Metric*. [online] Available at: https://www.sealights.io/software-development-metrics/10-powerful-agile-metrics-and-1-missing-metric/ [Accessed 24 Oct. 2019].

Anon, (2019). [online] Available at: https://www.atlassian.com/agile/project-management/metrics [Accessed 24 Oct. 2019].

phoenixNAP Blog. (2019). *15 KPIs & Metrics that Make the Case for DevOps*. [online] Available at: https://phoenixnap.com/blog/devops-metrics-kpis [Accessed 26 Oct. 2019].

Intellectsoft Blog. (2019). *13 Agile Metrics to Improve Productivity & Software Quality*. [online] Available at: https://www.intellectsoft.net/blog/agile-metrics/ [Accessed 26 Oct. 2019].

Perforce Software. (2019). *What Is Cyclomatic Complexity?* | *Perforce Software*. [online] Available at: https://www.perforce.com/blog/qac/what-cyclomatic-complexity [Accessed 26 Oct. 2019].

Intellectsoft Blog. (2019). *13 Agile Metrics to Improve Productivity & Software Quality*. [online] Available at: https://www.intellectsoft.net/blog/agile-metrics/ [Accessed 28 Oct. 2019].

GeeksforGeeks. (2019). *Software Engineering | Halstead's Software Metrics - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/ [Accessed 28. Oct 2019].

Saedsayad.com. (2019). *Bayesian Belief Network*. [online] Available at: https://www.saedsayad.com/docs/Bayesian_Belief_Network.pdf [Accessed 28 Oct. 2019].

Ptgmedia.pearsoncmg.com. (2019). [online] Available at: http://ptgmedia.pearsoncmg.com/images/0131424602/samplechapter/0131424602_ch05.pdf [Accessed 28 Oct. 2019].

GitPrime Help Center. (2019). *What Is GitPrime Exactly?*. [online] Available at: https://help.gitprime.com/general/what-is-gitprime-exactly [Accessed 29 Oct. 2019].

Humanyze. (2019). *Humanyze - Analytics For Better Performance.*. [online] Available at: https://www.humanyze.com/ [Accessed 29 Oct. 2019].

AIHR Analytics. (2019). *People Analytics: Ethical Considerations* | *AIHR Analytics*. [online] Available at: https://www.analyticsinhr.com/blog/people-analytics-ethical-considerations/ [Accessed 29 Oct. 2019].